UNIVERSITATEA DIN CRAIOVA
FACULTATEA DE ELECTROMECANICĂ
CATEDRA DE ACȚIONĂRI ELECTRICE

Şef lucrări dr. ing. Cătălin CONSTANTINESCU

BAZE DE DATE

Electromecanică - Frecvență redusă

- Suport teoretic -

2006 - 2007

BIBLIOGRAFIE

1. ***SISTEME DE GESTIUNE A BAZELOR DE DATE***
   Velicanu Manole şi colectiv
   Ed. Petrion, Bucureşti, 1998

2. ***BAZE DE DATE INTELIGENTE ÎN MANAGEMENTUL FIRMEI***
   Andone Ioan, Ţugui Alexandru
   Ed. Dosoftei, Iaşi, 1997

3. ***SQL FĂRĂ PROFESOR ÎN 14 ZILE***
   Perkins Jeff, Morgan Bryan
   Ed. TEORA, Bucureşti, 1998

4. ***BAZE DE DATE***
   Burdescu Dumitru Dan, Ionescu Augustin Iulian, Stănescu Liana
   Editura Universitaria, Craiova, 2004

5. ***BASES DE DONNÉES***
   Jean Fruitet
   Cours, Université de Marne - La - Vallée, 1997

6. ***PROGRAMARE ÎN ACCESS 97. VBA PENTRU ÎNCEPĂTORI***
   Smith Robert, Sussman David
   Ed. TEORA, Bucureşti, 1997

7. ***TOTUL DESPRE MICROSOFT ACCESS 97***
   Viescas L. John
   Ed. TEORA, Bucureşti, 1999

8. ***BAZE DE DATE. FUNDAMENTE TEORETICE ŞI PRACTICE***
   Grupul BDASEIG
   Ed. Infomega, Bucureşti, 2002

9. ***SQL în lecţii de 10 minute***
   Ben Forta
   Ed. Teora, Bucureşti, 2004

Baze de date şi sisteme de gestiune a bazelor de date. Elemente generale

O ***bază de date*** este un ansamblu organizat şi structurat de date legate funcţional între ele.

Bazele de date sunt gestionate unitar prin programe dedicate, numite ***sisteme de gestiune a bazelor de date*** (***SGBD***; ***DBMS*** - ***D***ata***B***ase ***M***anagement ***S***ystems).

Un SGBD (fig. 1) este responsabil de:

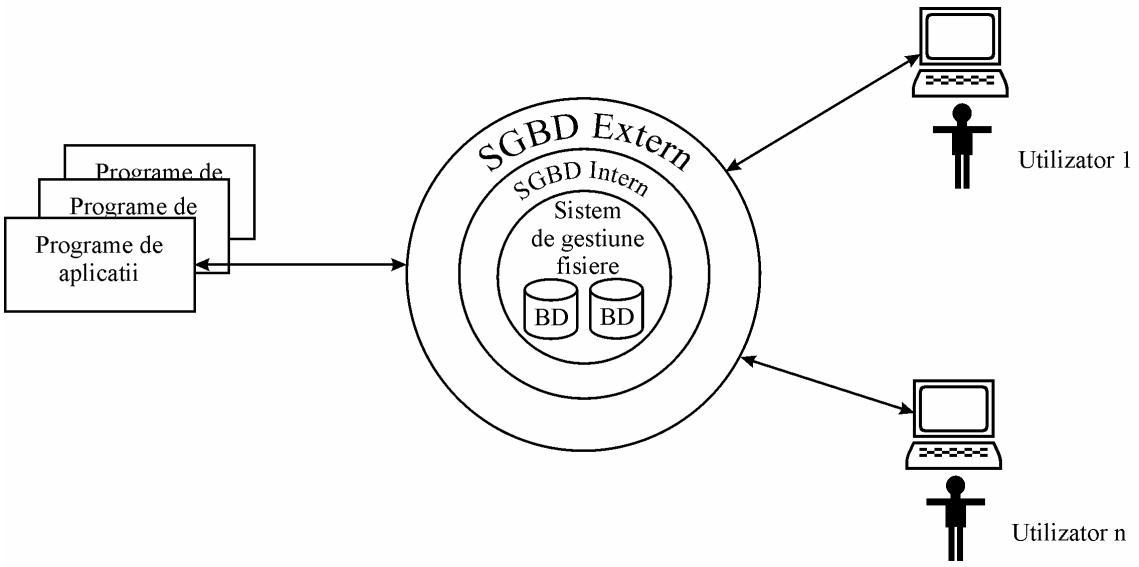


Fig. 1 Structura generală a unui sistem de gestiune a bazelor de date

- memorarea datelor prin intermediul sistemului de gestiune a fişierelor;
- gestiunea datelor şi a legăturilor dintre ele (SGBD intern);
- introducerea şi extragerea datelor (SGBD extern).

SGBD-urile actuale au 3 niveluri de reprezentare a datelor din bazele de date (fig. 2):

- nivelul extern care face referire la datele necesare utilizatorilor;
- nivelul conceptual care rezultă din nivelul extern prin analiza schemelor externe şi prin eliminarea redundanţelor;
- nivelul intern care se realizează cu ajutorul efectiv al SGBD-ului pornind de la schema conceptuală.

Funcţiile de bază ale unui SGBD sunt (fig. 3):

- funcţia de utilizare;
- funcţia de descriere a datelor;
- funcţia de manipulare;
- funcţia de administrare.

Funcţia de utilizare asigură interfaţa între utilizatori şi baza de date. Mulţimea utilizatorilor poate fi împărţită în trei categorii:

- programatorii de aplicaţii, care scriu programele ce utilizează baza de date;
- utilizatorii finali (beneficiarii propriu-zişi ai bazei de date), care, la rândul lor, pot fi împărţiţi în utilizatori experţi şi utilizatori neinformaticieni (aşa-numiţii utilizatori naivi);
- administratorul bazei de date care asigură administrarea unitară a bazei de date.

Funcţia de descriere a datelor se referă la definirea bazei de date cu ajutorul unui limbaj de descriere a datelor (LDD). Definirea datelor poate fi realizată la toate nivelurile SGBD-urilor (extern, conceptual, intern). În urma descrierii datelor, rezultă schema bazei de date.

Funcția de manipulare a datelor este cea mai complexă şi se materializează prin intermediul limbajelor de manipulare a datelor (LMD). La nivelul acestei funcții, datele sunt încărcate, actualizate, prelucrate şi regăsite.

Funcția de administrare cade în sarcina adminstratorului de rețea. Câteva dintre sarcinile acestuia sunt:

- organizarea bazei de date;
- realizarea schemei conceptuale;
- coordonarea proiectării bazei de date;
- autorizarea accesului la date;
- refacerea bazei de date în cazul alterării acestora etc.

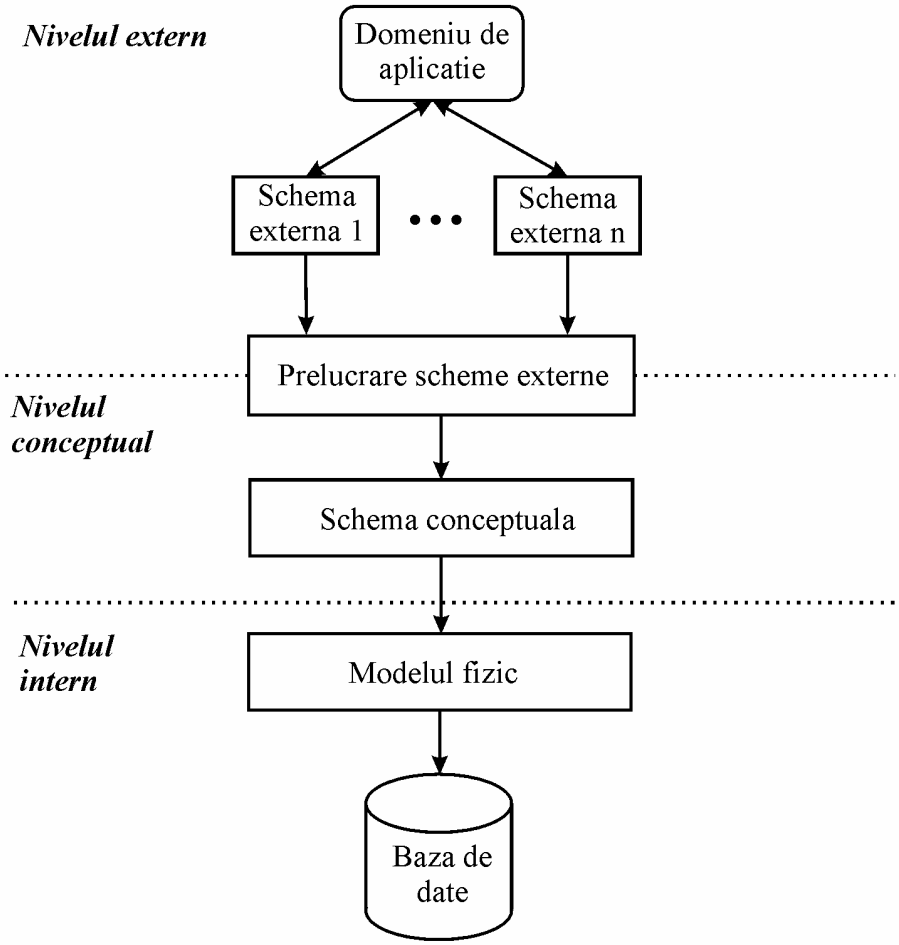


Fig. 2 Nivelurile de reprezentare şi percepție a SGBD-urilor

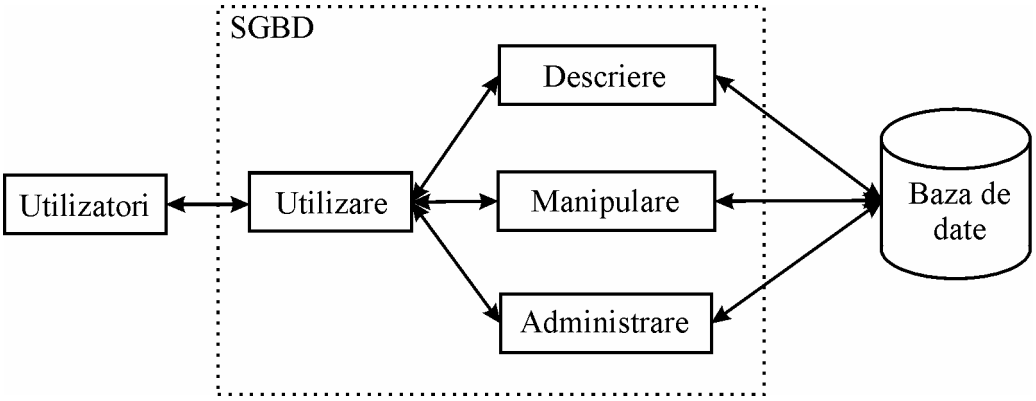


Fig. 3 Funcțiile de bază ale unui SGBD

Diagrama entitate-relație (diagrama ER)

O ***entitate*** este un obiect concret sau abstract care există şi poate fi distins de un alt obiect (de exemplu o persoană, un concept, un sentiment etc.).

O ***mulțime*** sau o ***clasă de entități*** este un grup de obiecte concrete sau abstracte de aceeaşi natură (de exemplu toate persoanele, toate conceptele, toate sentimentele etc.).

Un ***atribut*** reprezintă o proprietate caracteristică a entităților din aceeaşi clasă. De exemplu, o persoană poate fi caracterizată ptin următoarele caracteristici:

- nume;
- prenume;
- cod numeric personal (CNP);
- adresă;
- telefon.

Domeniul reprezintă mulțimea valorilor pe care le poate avea un anumit atribut.

O ***cheie primară*** (sau, mai simplu, o ***cheie***) este un atribut sau un set de atribute care identifică într-o manieră unică o entitate - de exemplu CNP (nu există două persoane care au acelaşi CNP).

Relația leagă între ele două sau mai multe entități prin intermediul unor atribute cu acelaşi domeniu. În cazul în care o relație leagă două entități, ea se numeşte ***binară***. Dacă relația este între cel puțin trei entități, se spune că este o ***relație n-ară***.

La rândul lor, relațiile binare se împart în trei categorii:

- relații ***unul la unul*** (***1:1***), caz în care unei entități E nu-i poate corespunde prin relația A decât o entitate F şi, invers, entității F nu îi poate corespunde decât entitatea E prin relația A;
- relații ***unul la mai mulți*** (***1:n***), caz în care unei entități E îi pot fi asociate mai multe entități F_i dar, unei entități F_i îi este asociată prin aceeaşi relație o singură entitate (E);
- relații ***mai mulți la mai mulți*** (***n:n***), caz în care unei entități E_i îi pot corespunde mai multe entități F_i şi reciproc.

Cardinalitatea unei perechi (E, A), unde E este o clasă de entități şi A este un set de relații, este dată de perechea (m, M), unde m şi M sunt numărul minim respectiv maxim de legături A ce leagă E de una sau mai multe clase de entități.

O ***diagramă ER*** este reprezentarea grafică a unei colecții de entități, relații, constrângeri, condiționări etc. care descriu complet o bază de date.

Exemplu:

Relația între un imobil şi un proprietar este de tipul "n la m", deoarece un imobil poate aparține mai multor proprietari şi, în acelaşi timp, un proprietar poate avea mai multe imobile ce aparțin aceleiaşi clase.

Fig. 4 Diagramă ER ce descrie dependența între un proprietar şi un imobil

Cardinalitățile sunt (0, n), respectiv (1,n), pentru că un imobil poate fi abandonat (deci poate să nu aibă nici un proprietar), dar, un proprietar trebuie să aibă cel puțin un imobil - dacă nu ar avea nici unul, nu s-ar mai numi proprietar.

În exemplul anterior sunt deja prezentate două simboluri grafice folosite în desenarea diagramelor ER - dreptunghiul corespunde unei entități, rombul corespunde unei relații şi

segmentele de dreaptă care fac legătura între entități şi relații. Alte două elemente grafice folosite sunt elipsa - prin care se reprezintă un atribut şi, din nou, segmentul de dreaptă, care poate face legătura şi între atribute şi entități.

Exemplul 1 - Să se deseneze diagrama ER a unei baze de date ce conține părinții unei persoane.

Atât părinții cât şi copii pot fi grupați într-o clasă de entități "PERSOANĂ". O entitate de acest tip poate fi descrisă prin atributele "NUME", "PRENUME", "SEX" şi "CNP". O relație ca cele din exemplul acesta se spune că este ***recursivă***.

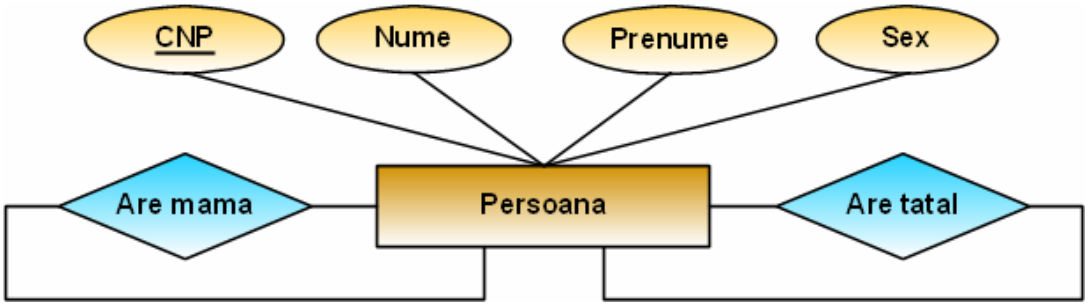


Fig. 5 Diagrama ER pentru exemplul 1

Obs. Atributul "CNP" este cheie primară. Pentru a se deosebi de celelalte atribute, în diagrama ER atributul se subliniază (fig. 4).

Exemplul 2 - Să se deseneze diagrama ER corespunzătoare unei baze de date care conține rețete culinare.

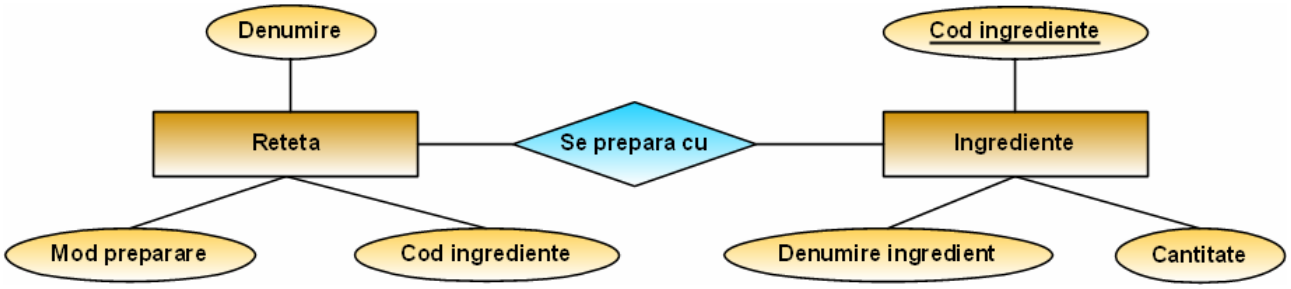


Fig. 6 Diagrama ER pentru exemplul 2

Pentru baza de date s-au imaginat două entități - "REȚETĂ" şi "INGREDIENTE", legătura dintre cele două entități făcundu-se, efectiv, prin atributele "COD INGREDIENTE", atribut care pentru entitatea "INGREDIENTE" este cheie primară.

Obs. Structurile bazelor de date pentru care s-au trasat diagramele entitate-relație nu sunt optimizate. Îmbunatatirea acestra va fi discutata ulterior.

Exemplul 3 - Să se deseneze diagrama ER a unei baze de date ce conține studenții unei facultăți.

Pentru această diagramă s-au imaginat trei entități legate funcțional între ele: entitatea "STUDENT", entitatea "DISCIPLINĂ" şi entitatea "NOTĂ".

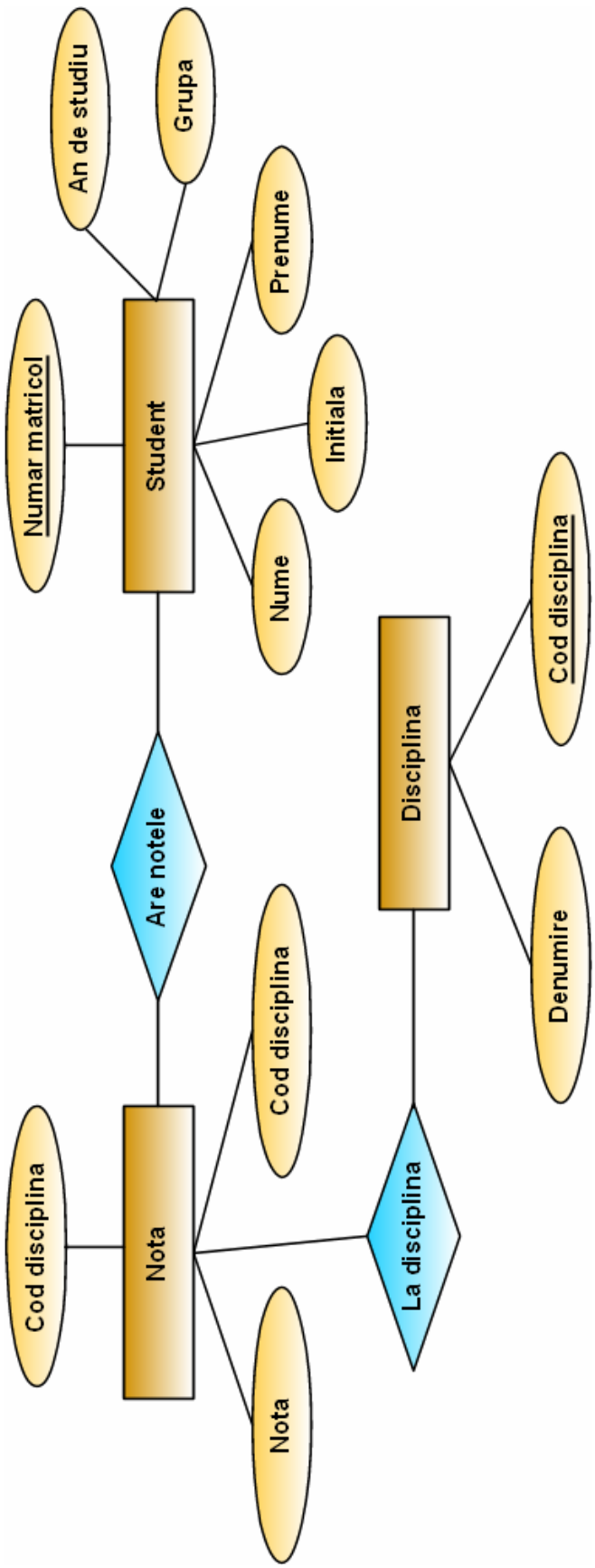


Fig. 7 Diagrama ER pentru exemplul 3

3. Normalizarea relațiilor

În procesul de modelare a bazelor de date, o etapa importantă o constituie ***normalizarea relațiilor***. Normalizarea presupune perfecționarea progresivă a schemei conceptuale ținând cont că trebuie asigurate următoarele cerințe:

- trebuie garantată ***conservarea datelor***;
- trebuie garantată ***conservarea dependențelor***;
- în urma normalizării, descompunerea relațiilor inițiale trebuie să fie una ***minimală***.

Necesitatea normalizării poate fi cel mai uşor evidențiată prin exemple de baze de date nenormalizate. În acest sens, în cele ce urmează se va comenta un mic exemplu de bază de date care gestionează notele unui student.

TNoteStudenti					
Nr_crt	Nr_matricol	Nume	Prenume	Disciplină	Notă
1	112	Ionescu	Bogdan	Baze de date	9
2	130	Vasilescu	Cristian	Aparate electrice	8
3	203	Popa	Virgil	Aparate electrice	6
4	112	Ionescu	Bogdan	Electronică digitală	10
5	203	Popa	Virgil	Electronică digitală	5
6	221	Florea	Ion	Baze de date	7
7	112	Ionescu	Bogdan	Baze de date	9

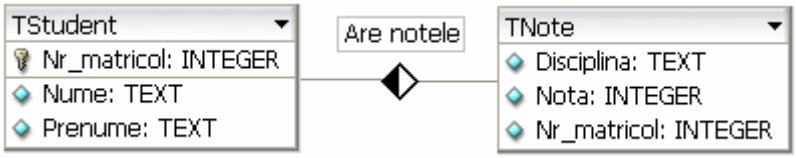
Datorită structurii acestei baze de date apar ca evidente cel puțin următoarele anomalii:

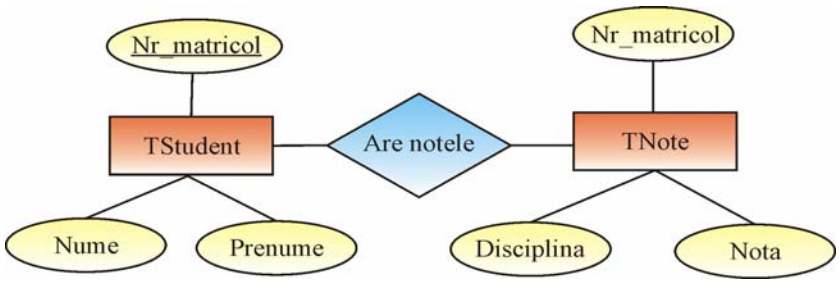
1. **Redundanță** – perechea de date (“Ionescu”, “Bogdan”) apare de 3 ori, (“Popa”, “Virgil”) de două ori etc.
2. **Anomalii la ştergere** – dacă se şterge înregistrarea cu Nr_crt = 6 se pierd toate informațiile legate de studentul “Florea Ion”.
3. **Anomalii la inserare** – perechea (“Ionescu”, “Bogdan”, “Baze de date”, 9) este inserată de două ori.
4. **Anomalii la modificare** – dacă prenumele studentului “Popa Virgil” este greşit, modificarea lui presupune modificarea unui număr de înregistrări egal cu notele studentului, ceea ce, între altele, poate conduce la noi erori de introducere a datelor.

Pentru eliminarea acestor anomalii prin normalizare, E.F. Codd a fundamentat o teorie matematică, care constă în esență în construirea unor tabele standard numite forme normale.

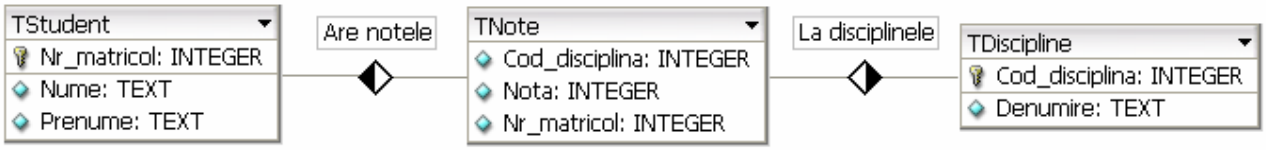
Până în prezent s-au dezvoltat mai multe astfel de forme, în practică fiind folosite numai o parte din acestea.

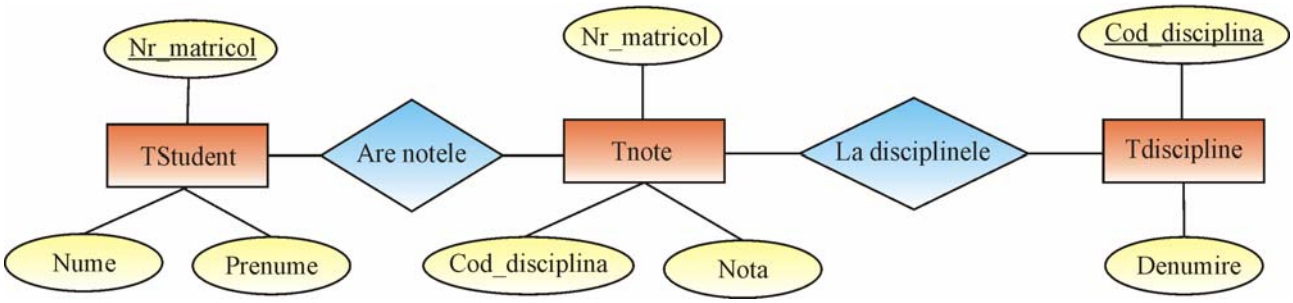
O structură a bazei de date discutate care elimină o parte din aceste anomalii “sparge” tabelul anterior în două tabele, conform următoarei reprezentări şi a diagramei entitate-relație asociate.





În această structură se observă, totuşi, o redundanţă a unor date, dată de apariţia repetată a numelor disciplinelor. Această redundanţă dispare prin introducerea în structură a unei noi entităţi, adică a unui nou tabel care să conţină denumirile disciplinelor.

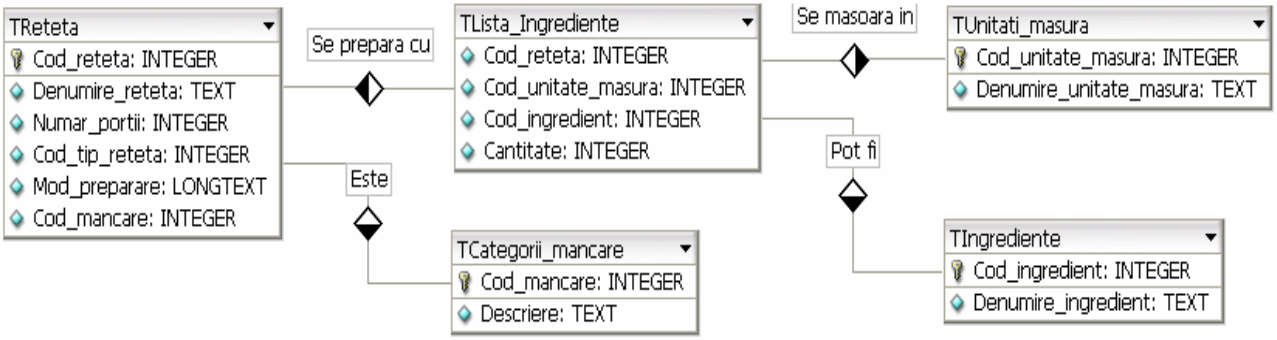




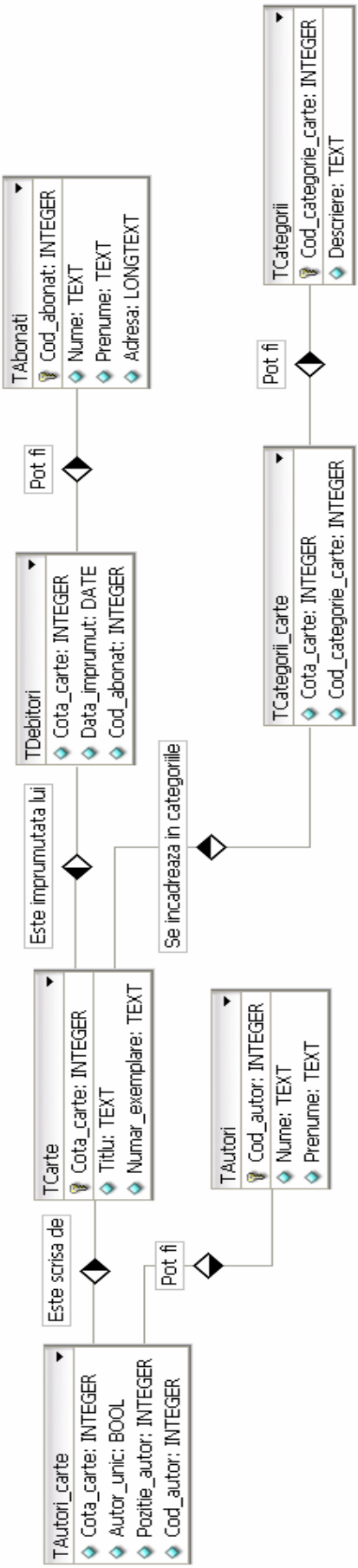
În scopul înţelegerii necesităţii nomalizării, se vor discuta în continuare alte exemple de baze de date.

Ex. 1. Structura unei baze de date care gestionează reţete culinare.

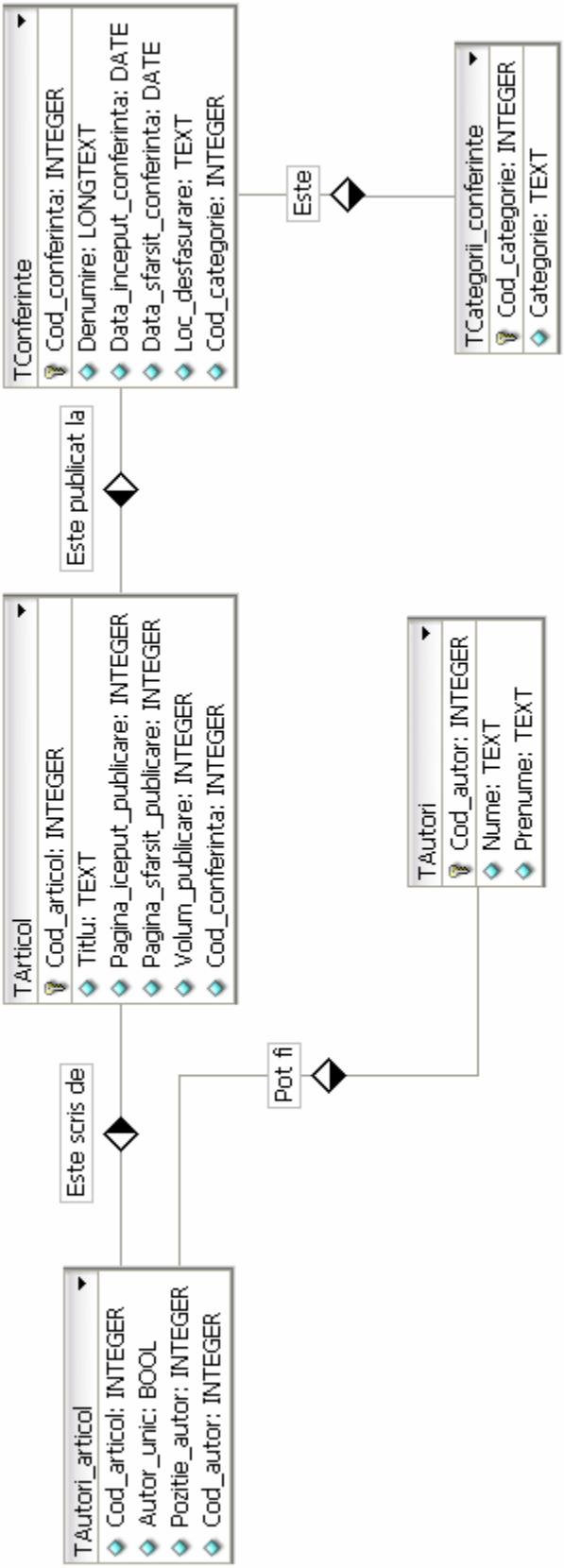
Pentru această baze de date, datele care este necesar a fi memorate sunt: denumirea reţetei, tipul acesteia (mâncare, desert, bautură etc.), ingredientele, cantităţile pentru un anumit număr de porţii, unităţile de măsură pentru acestea, numărul de porţii, modul de preparare.



Ex. 2. Structura unei baze de date care gestionează cărțile dintr-o bibliotecă publică



Ex. 3. Structura unei baze de date care gestionează articolele publicate de un grup de cercetători



4. Limbajul SQL (Structured Query Language)

Limbajul SQL se bazează pe interogări şi are ca obiect bazele de date relaţionale. Bazele de date relaţionale au fost introduse în 1970 de acelaşi cercetător de la IBM (E.F. Codd) care a pus bazele normalizării. Modelul bazelor de date relaţionale reprezintă baza de date ca o colecţie de tabele, fiecare dintre acestea relaţionând cu cel puţin un altul.

SQL este un limbaj prin intermediul căruia, utilizând comenzi derivate din limbajul natural, i se comunică unui SGBD ce se doreşte. Comenzile SQL permit, astfel, interogarea, inserarea, actualizarea şi ştergerea datelor.

SQL este referit uneori ca şi limbaj neprocedural. Asta înseamnă că nu este nevoie să se expliciteze locul unde SQL trebuie să caute un set de date, de exemplu. Este suficient să I se spună ce anume se doreşte. Localizarea informaţiei dorite în baza de date este sarcina SGBD-ului. În cazul limbajelor procedurale, fiecare interogare trebuie creată, compilată şi executată.

Sistemele de gestiune a bazelor de date comerciale utilizează SQL-ul în 2 nivele, astfel:

- Comenzile SQL se scriu direct ca şi linie de comandă, ceea ce face ca SGBD-ul să interpreteze imediat comenzile şi să furnizeze rezultatul. Această metodă de procesare se numeşte SQL interactiv.
- Cel de-al doilea nivel se numeşte SQL programat şi constă în integrarea comenzilor SQL într-un limbaj ca C, Basic, Cobol etc.

Aşa cum s-a mai spus, SQL operează cu baze de date relaţionale. Astfel, în mod firesc, un prim aspect care trebuie luat în discuţie este crearea bazelor de date.

Instrucţiunea CREATE TABLE

Sintaxa instrucţiunii CREATE TABLE este:

CREATE TABLE nume_tabel (câmp_1 tip_date_1, …, câmp_n tip_date_n);

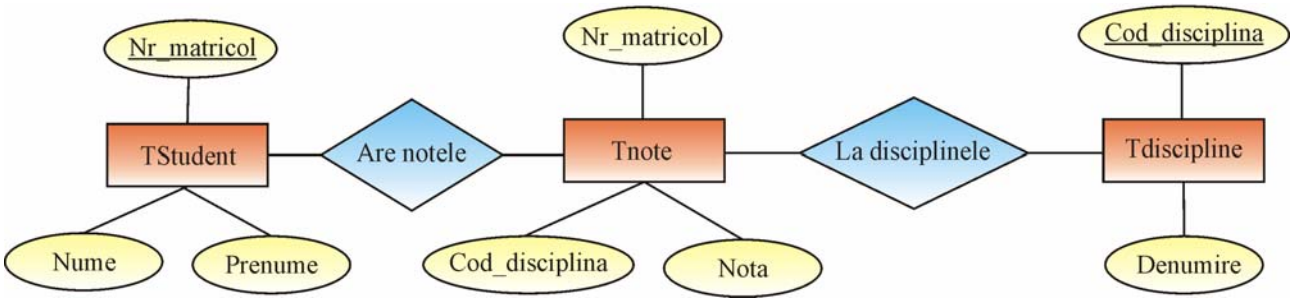
De exemplu, pentru crearea tabelului TnoteStudenţi (fig. de mai jos), codul SQL este:

TNoteStudenţi					
Nr_crt	Nr_matricol	Nume	Prenume	Disciplină	Notă
1	112	Ionescu	Bogdan	Baze de date	9
2	130	Vasilescu	Cristian	Aparate electrice	8
…					

```
CREATE TABLE TNoteStudenti (
     Nr_crt          INTEGER,
     Nr_matricol     INTEGER,
     Nume            CHAR(30),
     Prenume         CHAR(50),
     Disciplina      CHAR(100),
     Nota            INTEGER);
```

După tastarea acestei instrucţiuni, SGBD-ul va informa operatorul că tabelul a fost creat.

Este evident că instrucţiunea CREATE TABLE trebuie utilizată ori de câte ori se doreşte crearea unui nou tabel. Astfel, după normalizarea acestei baze de date, pentru că rezultă trei entităţi, adică trei tabele, codul SQL asociat definirii acestora este:




```
CREATE TABLE TStudent (
     Nr_matricol    INTEGER,
     Nume           CHAR(30),
     Prenume        CHAR(50));

CREATE TABLE Tnote (
     Nr_matricol    INTEGER,
     Cod_disciplina INTEGER,
     Nota           INTEGER);

CREATE TABLE Tdiscipline (
     Cod_disciplina INTEGER,
     Denumire       CHAR(100));
```

Obs. Comenzile SQL anterioare NU au declarat şi cheile primare ale entităţilor.

Introducerea datelor în tabele se face cu ajutorul instrucţiunii INSERT VALUES.

Instrucţionea INSERT VALUES

Sintaza instrucţiunii este:

INSERT INTO nume_tabel <(câmp_1, …, câmp_n)>
VALUES (valoare_câmp_1, …, valoare_câmp_n);

De exemplu inserarea celor două linii din tabelul TNoteStudenti se poate face cu următoarele comenzi SQL:

```
INSERT INTO TNoteStudenti (Nr_crt, Nr_matricol, Nume, Prenume, Disciplina, Nota)
      VALUES (1, 112, 'Ionescu', 'Bogdan', 'Baze de date', 9);

INSERT INTO TNoteStudenti
      VALUES (2, 130, 'Vasilescu', 'Cristian', 'Aparate electrice', 8);
```

Se observă pentru cea de-a doua înregistrare nu s-au mai specificat câmpurile tabelului. Acest lucru este posibil atunci când pentru o înregistrare se introdul toate câmpurile (adică nu există câmpuri vide nedeclarate) şi când ordinea datelor inserate în tabel este aceeaşi cu cea din definirea tabelului prin intermediul instrucţiunii CREATE TABLE.

Instrucţiunea SELECT

SELECT este utilizată pentru extragerea datelor din tabele.

În cazul în care se doreşte vizualizarea tuturor notelor studenţilor din tabelul TNoteStudenţi, se poate utiliza comanda:

```
SELECT Nume, Prenume, Disciplina, Nota FROM TNoteStudenti;
```

Din acest exemplu se poate observa uşor sintaxa instrucţiunii:

SELECT câmp_1, …, câmp_n FROM nume_tabel;

Obs. Pentru simplitate, toate instrucţiunile au fost prezentate până în acest moment în forma cea mai simplă. Dezvoltarea lor se va face progresiv, în discuţiile ulterioare.

Dacă se doreşte selecţia tuturor înregistrărilor dintr-un tabel, instrucţiunea SELECT se poate utiliza sub forma:

```
SELECT * FROM TNoteStudenti;
```

care este echivalentă cu

```
SELECT Nr_crt, Nr_matricol, Nume, Prenume, Disciplina, Nota
     FROM TNoteStudenti;
```

Instrucţiunea UPDATE

UPDATE permite modificarea înregistrărilor din baza de date.
Sintaxa instrucţiunii este:

```
UPDATE nume_tabel
     SET valoare_câmp_de_modificat = valoare
     WHERE criteriu_selectie_inregistrare;
```

De exemplu, dacă se doreşte modificarea notei studentului “Ionescu” din 9 în 10, codul SQL corespunzător este:

```
UPDATE TNoteStudenti
     SET Nota = 10
     WHERE Nume = 'Ionescu';
```

Deşi modificarea este corectă în tabelul dat exemplu (care conţine DOAR cele două înregistrări), în realitate execuţia acestei comenzi poate fi dezastruasă, datorită criteriului de selecţie. Astfel, după execuţie toţi studenţii cu numele “Ionescu” vor avea nota 10 la toate disciplinele. Comanda corectă este:

```
UPDATE TNoteStudenti
      SET Nota = 10
      WHERE Nume='Ionescu' AND Prenume='Bogdan' AND Disciplina='Baze de date';
```

sau, mai simplu:

```
UPDATE TNoteStudenti
      SET Nota = 10
      WHERE Nr_matricol = 112 AND Disciplina='Baze de date';
```

Crearea tabelelor virtuale

Tabelele create cu CREATE TABLE se numesc tabele de bază. SQL permite şi crearea unui alt tip de tabele, numite tabele virtuale sau vederi. În cazul tabelelor de bază atât structura tabelelor cât şi datele din acesta sunt memorate pe disc. Tabelele virtuale sunt derivate din tabelele de bază, astfel încât pe disc este memorată doar structura lor.

Pentru crearea vederilor se foloseşte instrucţiunea CREATE VIEW care conţine în corpul său şi o instrucţiune SELECT. De exemplu un tabel virtual ce conţine toate notele obţinute de studenţi se obţine cu următoare linie de cod SQL:

```
CREATE VIEW TNote
      AS SELECT NUME, PRENUME, DISCIPLINA, NOTA FROM TNoteStudenti
```

De fapt, o vedere este o interogare memorată pe disc. Ea este executată ori de câte ori este subiectul unei instrucţiuni şi poate fi referită ca orice tabel de bază. De exemplu aflarea notelor studentului "Ionescu" pot fi aflate astfel:

```
SELECT * FROM TNote WHERE NUME ='Ionescu' AND Prenume='Bogdan'
```

Crearea cheilor primare şi a cheilor externe

În exemplele discutate până acum, nu au fost specificate şi cheile dintr-un tabel, aşa cum s-a mai precizat. Acest lucru se face la nivelul instrucţiunii CREATE TABLE. De fapt, această instrucţiune permite specificarea unui număr mai mare de caracteristici ale coloanelor tabelelor. Pot fi specificate, de exemplu, coloanele nevide, sau a coloanelor ce conţini indecşi (valori unice).

De exemplu, crearea tabelului TNoteStudenţi poate fi făcută astfel:

```
CREATE TABLE TNoteStudenti (
     Nr_crt          INTEGER NOT NULL UNIQUE,
     Nr_matricol     INTEGER NOT NULL,
     Nume            CHAR(30),
     Prenume         CHAR(50),
     Disciplina      CHAR(100) NOT NULL,
     Nota            INTEGER NOT NULL);
```

Definirea tabelului în această manierăconduce la următoarele restricţii:

- coloanele Nr_crt, Nr_matricol, Disciplina şi Nota nu trebuie să fie vide într-o înregistrare;
- coloana Nr_crt nu poate avea aceeaşi valoare în două înregistrări diferite (în două linii ale tabelului).

Specificarea cheii primare se face cu modificatorul PRIMARY KEY. Acesta este relativ nou introdus în SQL, aşa că nu este cunoscu, deci acceptat, de toate sistemele. Cheia străină sau externă se declară prin modificatorul FOREIGN KEY.

Astfel, luând în considerare toate aspectele prezentate anterior, tabelele bazei de date normalizate ce conţine notele studenţilor pot fi definite astfel:

```
CREATE TABLE TStudent (
     Nr_matricol     INTEGER NOT NULL UNIQUE,
     Nume            CHAR(30) NOT NULL,
     Prenume         CHAR(50) NOT NULL,
     PRIMARY KEY (Nr_matricol));

CREATE TABLE Tdiscipline (
     Cod_disciplina INTEGER NOT NULL UNIQUE,
```

```
        Denumire        CHAR(100) NOT NULL,
        PRIMARY KEY (Cod_disciplina));

CREATE TABLE Tnote (
        Nr_matricol     INTEGER NOT NULL,
        Cod_disciplina INTEGER NOT NULL,
        Nota            INTEGER NOT NULL,
        FOREIGN KEY (Nr_matricol) REFERENCES TStudent,
        FOREIGN KEY (Cod_disciplina) REFERENCES Tdiscipline);
```

Schimbarea structurii tabelelor

Comanda ALTER TABLE permite modificarea structurii tabelelor. Astfel, cu ajutorul clauzei ADD pot fi adăugate noi coloane. Modificarea unor coloane deja existente se poate face cu ajutorul clauzei MODIFY. Mai mult, ştergerea coloanelor poate fi făcută prin intermediul clauzei DROP.

De exemplu, dacă se doreşte inserarea unui câmp care să conţină numărul de credite obţinute pentru un examen promovat, trebuie modificată structura tabelului TDiscipline.

Inserarea se face cu comanda ALTER TABLE astfel:

```
ALTER TABLE TDiscipline
        ADD (Nr_credite INTEGER);
```

Dacă se doreşte ştergerea acestei coloane, se foloseşte aceeaşi comandă combinată cu DROP:

```
ALTER TABLE TDiscipline DROP (Nr_credite);
```

Modificarea cu ALTER TABLE se poate face doar la nivelul stărilor UNIQUE şi NOT NULL. Pentru schimbări semnificative trebuie utilizată mai întâi clauza DROP (pentru ştergerea coloanei de modificat), apoi clauza ADD (pentru inserarea coloanei la care sunt luate în considerare toate modificările).

Modificarea stărilor UNIQUE şi / sau NOT NULL este indicat să se facă atunci când tabelul este gol, altfel pot apărea erori (o valoare NULL dintr-o înregistrare, de exemplu, poate intra în conflict cu declararea coloanei ca fiind nevidă prin NOT NULL).

Ştergerea tabelelor

Pentru ştergerea unui tabel dintr-o bază de date se foloseşte comanda DROP TABLE a cărui sintaxă este:

DROP TABLE nume_tabel;

În cazul unor SGBD-uri, tabelul trebuie să fie gol în momentul apelării comenzii DROP TABLE. Acestă caracteristică rezultă din motive de securitate, pentru prevenirea ştergerii accidentale a tabelelor.

Directiva DEFAULT

Pentru una sau pentru mai multe coloane pot fi definite valori implicite.

De obicei, o înregistrare pentru care nu s-a specificat un câmp, conţine în momentul salvării în baza de date valoarea NULL. Directiva DEFAULT suprascrie această valoare cu valoarea declarată prin intermediul ei.

Luînd în considerare atributul inserat anterior, adică numărul de credite, dacă se doreşte alocarea unui număr implicit de 5 credite corespunzător promovării unui examen, linia de cod SQL care permite acest lucru este:

```
CREATE TABLE Tdiscipline (
     Cod_disciplina INTEGER NOT NULL UNIQUE,
     Denumire       CHAR(100) NOT NULL,
     Nr_credite     INTEGER DEFAULT 5;
     PRIMARY KEY (Cod_disciplina));
```

Directiva CHECK

O parte din coloanele tabelelor trebuie să se încadreze într-un anumit domeniu sau să aibă un format particular. Directiva CHECK îi permite programatorului să specifice acest lucru. Revenind la baza de date ce gestionează notele studenților, pentru verificarea corectitudinii introducerii notei (în sensul că ea este cuprinsă în intervalul închis [1, 10]), declararea coloanelor tabelului în SQL devine:

```
CREATE TABLE Tnote (
     Nr_matricol    INTEGER NOT NULL,
     Cod_disciplina INTEGER NOT NULL,
     Nota           INTEGER NOT NULL
          CHECK (Nota > 0 AND Nota <= 10),
     FOREIGN KEY (Nr_matricol) REFERENCES TStudent,
     FOREIGN KEY (Cod_disciplina) REFERENCES TDiscipline);
```

Interogări complexe

În exemplele anterioare comanda SELECT s-a utilizat doar pentru obținerea unor interogări directe, simple. SELECT permite, însă, şi ordonarea datelor, specificarea unor operații aritmetice şi / sau logice ce se efectuează asupra datelor obținute, permite specificarea unor criterii de selecție a datelor ce se vor obține după interogare etc.

Astfel sintaxa completă a comenzii SELECT este:

```
SELECT [DISTINCT] câmp_expresie [,câmp_expresie]
FROM nume_tabel [, nume_tabel]
     [WHERE condiție_selecție]
     [ORDER BY nume_câmp [, nume_câmp]]
     [GROUP BY nume_câmp [, nume_câmp]]
     [HAVING condiție];
```

unde:

- *câmp_expresie* poate fi un câmp (un atribut) al unui tabel, una dintre funcțiile standard de agregare (SUM – sumă, AVG – medie, MIN – minim, MAX – maxim, COUNT – numărare), sau, aşa cum s-a mai spus, caracterul * pentru selecția tuturor atributelor;
- *nume_tabel* este numele tabelului care intervine în interogare;
- *condiție_selecție* specifică care înregistrări se doreşte a fi obținute în urma interogării;
- *nume_câmp* reprezintă un câmp, o coloana a unui tabel implicat în interogare;
- câmpul *condiție* din cadrul clauzei HAVING permite filtrarea unui grup de înregistrări.

Pentru exemplificare se va considera tot baza de date care gestionează notele studenților.

Dacă se doreşte determinarea mediei studentului cu numărul matricol 123, interogarea va fi de forma:

```
SELECT AVG(Nota) FROM Tnote
     WHERE Nr_matricol = 123;
```

Dacă se doreşte determinarea numărului de studenţi care au promovat examenul la disciplina al cărui cod este 15, de exemplu, interogarea devine:

```
SELECT COUNT(*) FROM Tnote
     WHERE Cod_disciplina = 15 AND Nota >= 5;
```

Pentru obţinerea tuturor studenţilor în ordine alfabetică comanda SELECT se utilizează împreună cu clauza ORDER BY, astfel:

```
SELECT * FROM TStudent
     ORDER BY Nume, Prenume;
```

GROUP BY se foloseşte pentru obţinerea unor grupuri în cadrul interogărilor obţinute cu ajutorul instrucţiunii SELECT.

De exemplu, cu linia de cod:

```
SELECT Nr_matricol, COUNT(*) AS Examene_promovate FROM Tnote
     WHERE Nota >= 5
     GROUP BY Nr_matricol;
```

are ca efect afişarea numărului de examene promovate de studenţii din baza de date, identificaţi în acest caz prin numărul matricol, după care se face şi gruparea.

Directiva AS alocă numele formal Examene_promovate sumei examenelor la care un student a obţinut cel puţin nota 5.

HAVING permite specificarea unei condiţii de filtrare a datelor.

Dacă se doreşte selecţia doar a studenţilor care au cel puţin 5 examene promovate, codul SQL aferent este:

```
SELECT Nr_matricol, COUNT(*) AS Examene_promovate FROM Tnote
     WHERE Nota >= 5
     GROUP BY Nr_matricol
     HAVING COUNT(*) >= 5;
```

Clauza DISTINCT este foarte utilă atunci când este necesară aflarea tuturor valorilor unui atribut, nefiind important numărul de apariţii.

De exemplu, dacă se doreşte afişarea tuturor numerelor matricole alocate, se poate utiliza comanda SELECT astfel:

```
SELECT DISTINCT Nr_matricol FROM Tstudent;
```

În condiţiile de selecţie din exemplele anterioare au intervenit câţiva operatori de comparare. Mulţimea acestor operatori conţine următoarele simboluri:

- = - egal;
- > - mai mare;
- < - mai mic;
- >= - mai mare sau egal decât;
- <= - mai mic sau egal cu;
- <> - diferit.

Pentru verificarea încadrării valorii unui atribut într-un anumit interval poate fi folosit operatorul BETWEEN. Dacă se doreşte selecţia tuturor studenţilor cu mediile cuprinse între 7,50 şi 9,50 dintr-o vedere ce conţine mediile acestora comenzile SQL necesare sunt:

Creare vedere:

```
CREATE VIEW VMedii_studenti AS
SELECT Nr_matricol, AVG(Nota) AS Medie FROM Tnote
     GROUP BY Nr_matricol;
```

Interogare (cu utilizarea operatorului BETWEEN):

```
SELECT * FROM VMedii_studenti
     WHERE Medie BETWEEN 7.50 AND 9.50;
```

În cazul în care se doreşte obţinerea numelui studenţilor cu mediile cuprinse între cele două limite (nu numai numărul matricol), în corpul comenzii select se vor specifica atât numele vederii cât şi numele tabelului care conţine numele şi prenumele studenţilor:

```
SELECT VMedii_studenti.Nr_matricol, TStudent.Nume, TSudent.Prenume, VMedii_studenti.Medie
FROM VMedii_studenti, TStudent
     WHERE VMedii_studenti.Medie BETWEEN 7.50 AND 9.50,
           TStudent.Nr_matricol = VMedii_studenti.Nr_matricol;
```

sau

```
SELECT VMS.Nr_matricol, TS.Nume, TS.Prenume, VMS.Medie
FROM VMedii_studenti VMS, Tstudent TS
     WHERE VMS.Medie BETWEEN 7.50 AND 9.50,
          TS.Nr_matricol = VMS.Nr_matricol;
```

În cel de-al doilea caz, pentru fiecare dintre tabelele implicate în interogare s-a asociat câte un nume formal: VMS pentru tabelul virtual Vmedii_studenţi şi TS pentru tabelul TStudent. Acest lucru determină simplificarea modului de scriere a unei interogării.

Pentru recapitularea şi fixarea elementelor de limbaj SQL prezentate până acum, se va construi în cele ce urmează o bază de date care gestionează articolele dintr-un raion de echipamente electrocasnice.

Crearea tabelelor care conţin chei primare şi nu conţin chei străine:

```
CREATE TABLE TProducatori (
     Cod_producator INTEGER NOT NULL UNIQUE,
     Denumire       CHAR(200) NOT NULL,
     PRIMARY KEY (Cod_producator));

CREATE TABLE TFirme (
     Cod_firma      INTEGER NOT NULL UNIQUE,
     Nume           CHAR(100) NOT NULL,
     Adresa         CHAR(200) NOT NULL,
     PRIMARY KEY (Cod_firma));
```

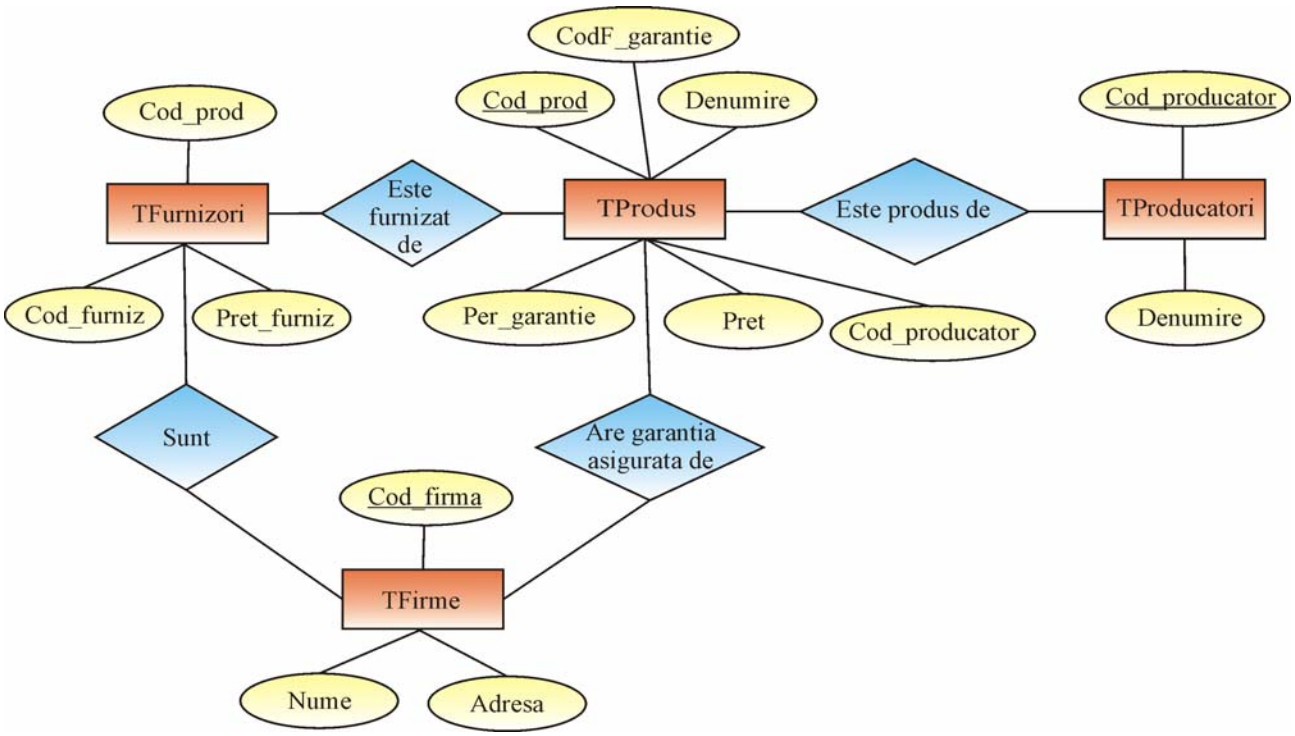


Diagrama entitate-relație pentru o bază de date care gestionează echipamentele electrocasnice dintr-un raion specializat

Crearea tabelelor care conțin atât chei primare cât şi chei străine:

```
CREATE TABLE TProdus (
     Cod_prod        INTEGER NOT NULL UNIQUE,
     Denumire        CHAR(100) NOT NULL,
     Pret            INTEGER NOT NULL,
     Cod_producator  INTEGER,
     Per_garantie    INTEGER DEFAULT 1,
     CodF_garantie   INTEGER NOT NULL,
     PRIMARY KEY (Cod_prod),
     FOREIGN KEY (Cod_producator) REFERENCES TProducatori,
     FOREIGN KEY (CodF_garantie) REFERENCES TFirme);
```

Crearea tabelelor care conțin numai chei străine:

```
CREATE TABLE TFurnizori (
     Cod_prod        INTEGER NOT NULL,
     Cod_furniz      INTEGER NOT NULL,
     Pret_furniz     INTEGER NOT NULL,
     FOREIGN KEY (Cod_prod) REFERENCES TProdus,
     FOREIGN KEY (Cod_furniz) REFERENCES TFirme);
```

Inserarea a două înregistrări ce conțin date referitoare la firme furnizoare noi:

```
INSERT INTO TFirme (Nume, Cod_firma, Adresa) VALUES ('SC X SRL', 16, 'Adresa_X');
INSERT INTO TFirme VALUES (17, 'SC Y SRL', 'Adresa_Y');
```

Crearea unei interogări care să extragă toate echipamentele produse de firma ‘Z’.

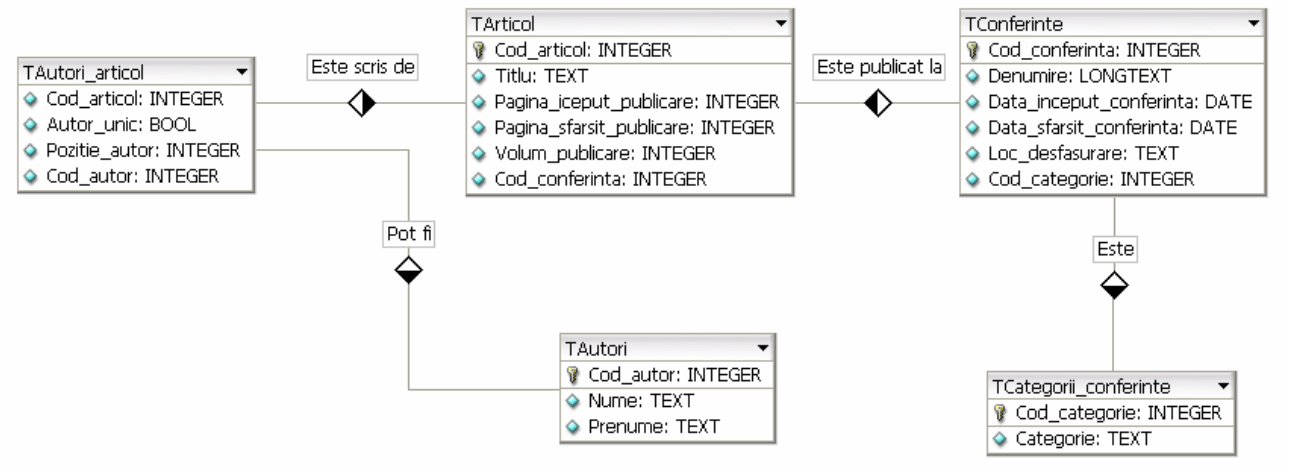
```
SELECT A.Cod_prod, A.Denumire, A.Pret
FROM Tprodus A, TProducatori B
     WHERE A.Cod_producator = B.Cod_producator, B.Denumire = 'Z';
```

5. Aplicații specifice bazelor de date

Limbajul SQL este practic integrat în toate SGBD-urile actuale şi nu numai. Instrucțiunile SQL pot fi utilizate fără probleme şi în limbaje de nivel înalt precum Pascal (Delphi), Basic (Visual Basic), C (Visual C) etc. În ideea simplificării modulului de utilizare a scripturilor SQL s-a dezvoltat o gama largă de aplicații, multe dintre ele fiind distribuite gratuit. În capitolul de față se vor preznta doar câteva exemple de aplicații de acest gen, tocmai pentru a se pune în evidență utilitatea acestora.

O etapa importantă în dezvoltarea unei baze de date o constituie stabilirea structurii acesteia, structură care poate fi descrisa complet prin intermediul diagramei entitate-relație. Odată trasată această diagramă, dacă se utilizează aplicții specifice, poate fi generat codul SQL responsabil de crearea tabelelor şi a legăturilor dintre acestea.

Ex. 1 - Pentru acest prim exemplu se va considera DER trasată pentru baza de date care gestionează articolele publicate de un grup de cercetători.

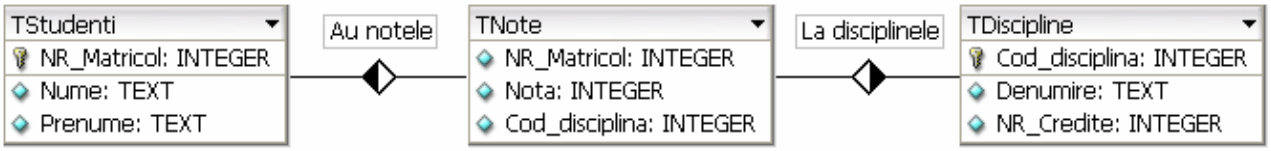


Codul SQL generat cu o aplicație specifică:

```
CREATE TABLE TArticol (
  Cod_articol INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  Titlu TEXT NULL,
  Pagina_iceput_publicare INTEGER UNSIGNED NULL,
  Pagina_sfarsit_publicare INTEGER UNSIGNED NULL,
  Volum_publicare INTEGER UNSIGNED NULL,
  Cod_conferinta INTEGER UNSIGNED NULL,
  PRIMARY KEY(Cod_articol)
);

CREATE TABLE TAutori (
  Cod_autor INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  Nume TEXT NULL,
  Prenume TEXT NULL,
  PRIMARY KEY(Cod_autor)
);

CREATE TABLE TAutori_articol (
  Cod_articol INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  Autor_unic BOOL NULL,
  Pozitie_autor INTEGER UNSIGNED NULL,
  Cod_autor INTEGER UNSIGNED NULL
);
```

```
CREATE TABLE TCategorii_conferinte (
  Cod_categorie INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  Categorie TEXT NULL,
  PRIMARY KEY(Cod_categorie)
);

CREATE TABLE TConferinte (
  Cod_conferinta INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  Denumire LONGTEXT NULL,
  Data_inceput_conferinta DATE NULL,
  Data_sfarsit_conferinta DATE NULL,
  Loc_desfasurare TEXT NULL,
  Cod_categorie INTEGER UNSIGNED NULL,
  PRIMARY KEY(Cod_conferinta)
);
```

Ex. 2 - Definirea tabelelor pentru o bază de date care gestionează notele studenților unei facultăți.

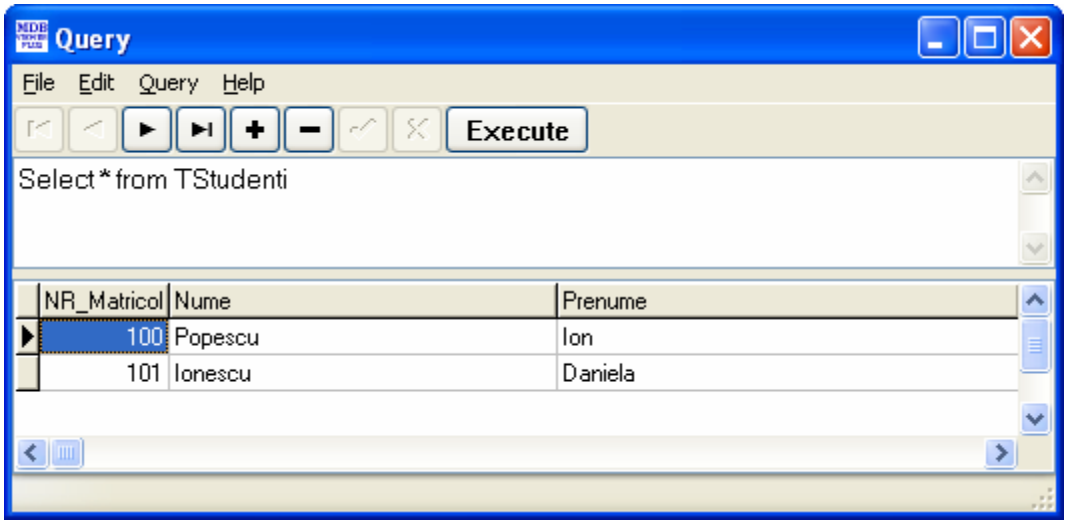



```
CREATE TABLE TDiscipline (
  Cod_disciplina INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  Denumire TEXT NULL,
  NR_Credite INTEGER UNSIGNED NULL,
  PRIMARY KEY(Cod_disciplina)
);

CREATE TABLE TNote (
  NR_Matricol INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  Nota INTEGER UNSIGNED NULL,
  Cod_disciplina INTEGER UNSIGNED NULL
);

CREATE TABLE TStudenti (
  NR_Matricol INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  Nume TEXT NULL,
  Prenume TEXT NULL,
  PRIMARY KEY(NR_Matricol)
);
```

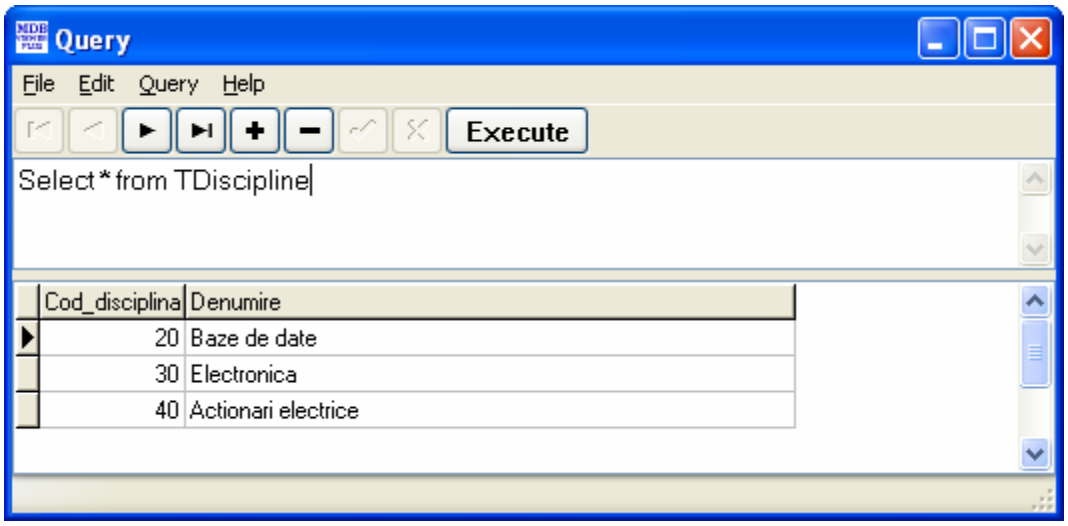
Sunt, de asemenea, dezvoltate aplicații care permit utilizarea instructiunilor SQL pentru extragerea directă a datelor din baze de date în format dBase, Microsoft Access, Paradox etc.

Ex. 3 - Utilizarea instrucțiunii SELECT pentru extragerea datelor dintr-o bază de date ce gestionează notele studenților unei facultăți.

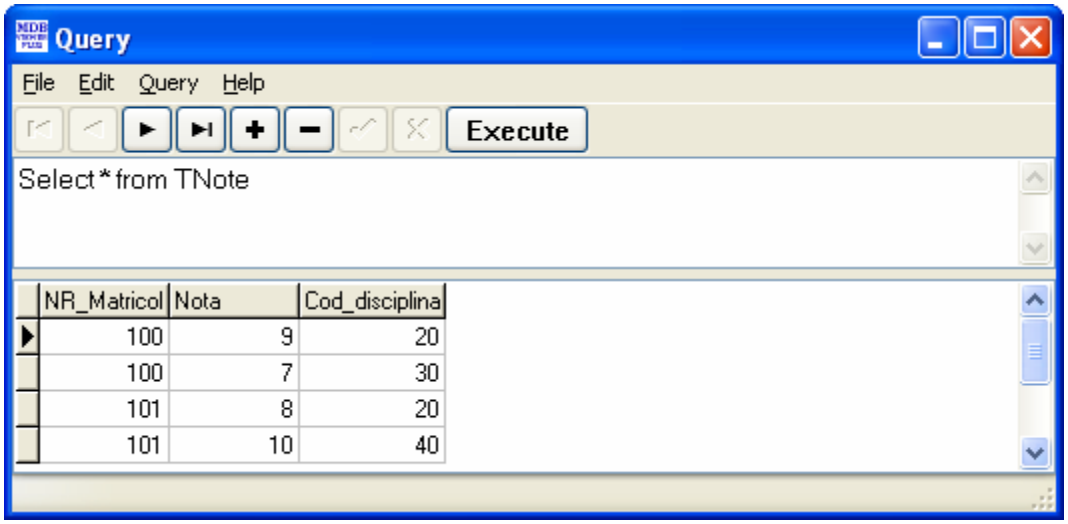
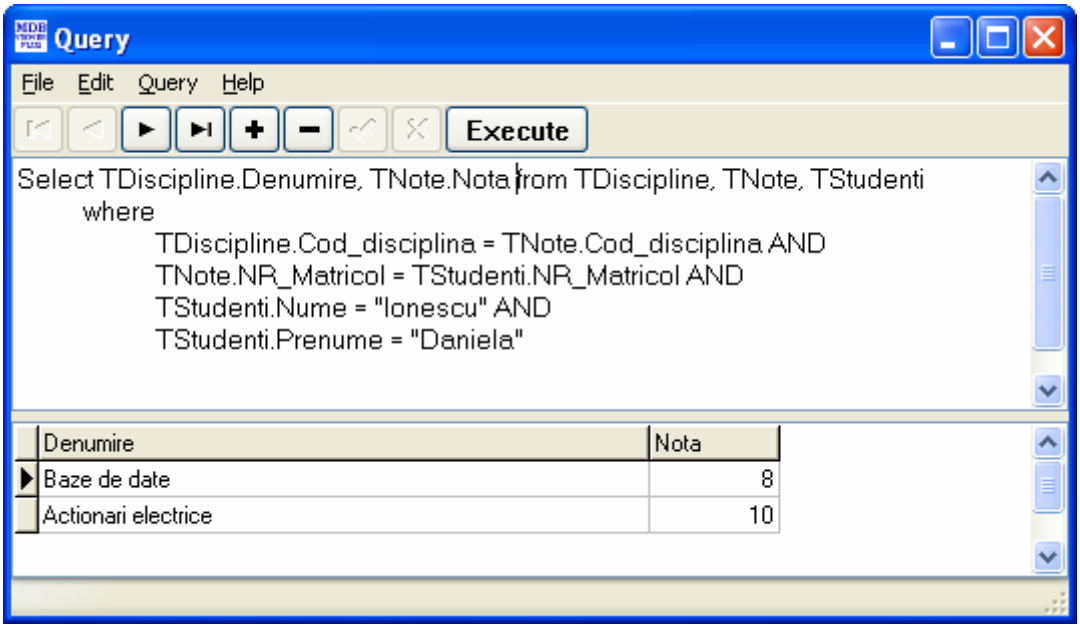
Selecția tuturor înregistrărilor tabelului **TStudenti**.



Selecția tuturor înregistrărilor tabelului **TDiscipline**.



Selecția tuturor înregistrărilor tabelului **TNote**.



Selecția tuturor notelor studentei “Ionescu Daniela”.